Chapter 2

Residue Numbers and the Limits of Fast Arithmetic

In this chapter, we are concerned about the speed of arithmetic. In most arithmetic systems, the speed is limited by the nature of the building block that makes logic decisions and the extent to which decisions of low order numeric significance can affect results of higher significance. This latter problem is best illustrated by the addition operation, in which a low order carry can have a rippling effect on a sum.

We begin by examining ways of representing numbers, especially insofar as they can reduce the sequential effect of carries on digits of higher significance. Carry independent arithmetic (called residue arithmetic) is possible within some limits. This residue arithmetic representation is a way of approaching a famous bound on the speed at which addition and multiplication can be performed.

This bound, called Winograd's bound, determines a minimum time for arithmetic operations and is an important basis for determining the comparative value of various implementation algorithms to be discussed in subsequent chapters.

For certain operations storage, especially a Read Only Memory (ROM), can be used to "look-up" a result or partial result. Since very dense ROM technology is now available, the last section of this chapter develops a performance model of ROM access. Unlike Winograd's work, this is not a strict bound, but rather an approximation to the retrieval time.

2.1 The Residue Number System

2.1.1 Representation

The number systems considered in the last chapter are linear, positional, and weighted, in which all positions derive their weight from the same radix (base). In the binary number systems, the weights of the positions are 2^0, 2^1, 2^2, etc. In the decimal number system, the weights are $10^0 = 1$, $10^1 = 10$, $10^2 = 100$, $10^3 = 1000$, etc.

The residue number system [15, 20] usually uses positional bases that are relatively prime to each other, for example, 2, 3, 5. For instance, if the number 8 is divided by the base 5, the residue is 3. The following table lists the numbers 0 to 29 and their residues to bases 2, 3, and 5. (The number of unique representations is $2 \times 3 \times 5 = 30$.)

	Residue to base				Residue to base				Residue to base		
N	5	3	2	N	5	3	2	N	5	3	2
0	0	0	0	10	0	1	0	20	0	2	0
1	1	1	1	11	1	2	1	21	1	0	1
2	2	2	0	12	2	0	0	22	2	1	0
3	3	0	1	13	3	1	1	23	3	2	1
4	4	1	0	14	4	2	0	24	4	0	0
5	0	2	1	15	0	0	1	25	0	1	1
6	1	0	0	16	1	1	0	26	1	2	0
7	2	1	1	17	2	2	1	27	2	0	1
8	3	2	0	18	3	0	0	28	3	1	0
9	4	0	1	19	4	1	1	29	4	2	1

The residues in the above table uniquely identify a number. The configuration $[2, 1, 1]$ represents the decimal number 7 just as uniquely as binary 111.

To convert a conventionally weighted number (X) to the residue system, we simply take the residue of X with respect to each of the positional moduli.

Example 2.1

To convert the decimal number 29 to a residue number, we compute:

$$\begin{aligned} R_5 &= 29\,mod\; 5 = 4 \\ R_3 &= 29\,mod\; 3 = 2 \\ R_2 &= 29\,mod\; 2 = 1 \end{aligned}$$

The decimal number 29 is represented by $[4, 2, 1]$ in the above residue number system. $\diamondsuit$

The main advantage of the residue number system is the absence of carries between columns in addition and in multiplication. Arithmetic is closed (done completely) within each residue position. Therefore, it is possible to perform addition and multiplication on long numbers at the same speed as on short numbers, since the speed is determined by the largest modulus position. Recall that in the conventional linear weighted number system, an operation on long words is slower due to the carry propagation.

2.1.2 Operations in the Residue Number System

Examples of additions in 5, 3, 2 residue arithmetic are:

$$\begin{array}{rcl@{\qquad}rcl} 9 & \rightarrow & [4,0,1] & 8 & \rightarrow & [3,2,0] \\ \underline{+16} & \rightarrow & \underline{[1,1,0]} & \underline{+19} & \rightarrow & \underline{[4,1,1]} \\ 25 & \rightarrow & [0,1,1] & 27 & \rightarrow & [2,0,1] \\ \text{decimal} & & \text{residue} & \text{decimal} & & \text{residue} \\ & & 5,3,2 & & & 5,3,2 \end{array}$$

Note that each column was added modulo its base, disregarding any interposition carries. An example of multiplication is:

$$\begin{array}{rcr} 7 & \rightarrow & [2,1,1] \\ \underline{\times 4} & \rightarrow & \underline{\times[4,1,0]} \\ 28 & & [3,1,0] \end{array}$$

Again, each column is multiplied modulo its base, disregarding any interposition carries; for example, $2 \times 4\, mod\ 5 = 8\, mod\ 5 = 3$.

The uniqueness property is the result of the famous Chinese Remainder Theorem.

Theorem 1 Chinese Remainder

Given a set of relatively prime moduli $(m_1, m_2, \ldots, m_i, \ldots, m_n)$, then for any $X < M$, the set of residues $\{X\ mod\ m_i | 1 \leq i \leq n\}$ is unique, where

$$M = \prod_{i=1}^{n} m_i.$$

The proof is straightforward. Suppose there were two numbers Y and Z that have identical residue representations; i.e., for each i, $y_i = z_i$, where

$$\begin{aligned} y_i &= Y \bmod m_i \\ z_i &= Z \bmod m_i. \end{aligned}$$

Then $Y - Z$ is a multiple of m_i, and $Y - Z$ is a multiple of the least common multiple of m_i. But since the m_i are relatively prime, their least common multiple is M. Thus, $Y - Z$ is a multiple of M, and Y and Z cannot both be less than M [37].

Subtraction: Since $(a\, mod\ m) - (b\, mod\ m) = (a - b)\, mod\ m$, the subtraction operation poses no problem in residue arithmetic, but the representation of negative numbers requires the use of complement coding.

Following our earlier discussion on complementation, we create a signed residue system by designating numbers $X < M/2$ as positive, and $X \geq M/2$ as negative, for all $X < M$. That is, a number $X \geq M/2$ is treated as $X - M$,

$$(X - M)\, mod\ M = X\, mod\ M,$$

and the complement of $X\, mod\ M$ is:

$$X^c = (M - X)\, mod\ M.$$

In residue representation $X = [x_i]$, where $x_i = X\, mod\, m_i$ and the complement of X is the complement of $[x_i]$. Call the complement of X, X^c and of x_i, x_i^c. Then

$$x_i^c = (m_i - x_i)\, mod\ m_i$$

and

$$[x_i]^c = [x_i^c] = X^c,$$

since $-x_i$ and $m_i - x_i$ are congruent, mod m_i.

Example 2.2

In the 5,3,2 residue system, $M = 30$, integer representations 0 through 14 are positive, and 15 through 29 are negative (i.e., represent numbers -15 through -1). Now:

$$8 = [3, 2, 0],$$

$$9 = [4, 0, 1],$$

$$(8)^c = [2, 1, 0] \quad \text{i.e., } 5 - 3, 3 - 2, \text{ and } (2 - 0)\ mod\ 2,$$

and

$$(9)^c = [1, 0, 1].$$

$$\begin{array}{rcccrl} 8 & = & 8 & = & [3,2,0] & \\ \underline{-9} & = & (9)^c & = & \underline{+[1,0,1]} & \\ -1 & & & & [4,2,1] & = 29 \text{ or } -1 \end{array}$$

◊

2.1.3 Selection of the Moduli

Certain moduli are more attractive than others for two reasons:

1. They are efficient in their binary representation; that is, n binary bits can represent approximately 2^n distinct residues.
2. They provide straightforward computational operations using binary adder logic.

Moduli of the form 2^{k_1}, $2^{k_1} - 1$, $2^{k_2} - 1$, $\ldots 2^{k_n} - 1$ ($k_1, k_2, \ldots, k_n$ are integers) were suggested by Merrill [26] as meeting the above criteria.

Note that not all numbers of the form $2^k - 1$ are relatively prime. In fact, if k is even:

$$2^k - 1 = (2^{k/2} - 1)(2^{k/2} + 1).$$

If k is an odd composite, $2^k - 1$ is also factorable, and for $k = p$, with p a prime, the resulting numbers may or may not be prime. For $k = a \cdot b$, the factors of $2^k - 1$ are $(2^a - 1)$ and $\left(2^{a(b-1)} + 2^{a(b-2)} + \ldots + 2^{a(0)}\right)$, whose product is $\left(2^{a \cdot b} - 1\right) = (2^k - 1)$. For $k = p$, a prime, we have the famous Merseene's numbers [27]:

$$M_p = 2^p - 1 \qquad (p \text{ a prime}).$$

Table 2.1: A Partial List of Moduli of the Form 2^k and $2^k - 1$ and Their Prime Factors

Moduli	Prime Factors
3	—
7	—
15	3,5
31	—
63	3,7
127	—
255	3,5
511	7,73
1023	3,11,31
2047	23,89
4095	3,5,7,13
8191	—
2^k $(k = 1, 2, 3, 4 \ldots)$	2

Merseene asserted in 1644 that the only p's for which M_p is prime are:

$$p = 2, 3, 5, 7, 13, 17, 19, 31, 67, 127, 257.$$

The conjecture stood for almost 300 years. In a historic paper in 1903, F. N. Cole showed that M_{67} was not a prime.

Table 2.1 lists factors for numbers of the form $2^k - 1$. Note that any 2^n will be relatively prime to any $2^k - 1$. The table is from Merrill [26].

Since the addition time is limited in the residue system to the time for addition in the largest module, we should select moduli as close as possible to limit the size of the largest modulus. Merrill suggests the largest be of the form 2^k and the second largest of the form $2^k - 1$, k the same. The remaining moduli should avoid common factors. He cites some examples of interest:

Bits to represent	Moduli set
17	32, 31, 15, 7
25	128, 127, 63, 31
28	256, 255, 127, 31

If the moduli are relatively prime, we can "almost" represent as many objects as the pure binary representation. For example, in the 17-bit case, instead of 2^{17} code points, we have

$$2^5(2^5 - 1)(2^4 - 1)(2^3 - 1) = 2^{17} - \mathcal{O}(2^{14}).$$

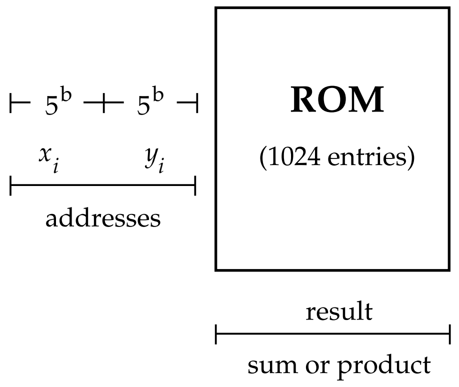
where $\mathcal{O}(2^{14})$ indicates a term on the order of 2^{14}. Thus, we have lost less than 1 bit of representational capability (a loss of 1 bit would correspond to an $\mathcal{O}(2^{16})$ loss).

2.1.4 Operations with General Moduli

With the increasing availability of ROM (Read Only Memory) technology, the restriction to moduli forms 2^k or $2^k - 1$ is less important. Thus, addition, subtraction, and multiplication can be done by table look-up. In the most straightforward implementation, separate tables are kept for each modulus, and the arguments x_i and y_i (both mod m_i) are concatenated to form an address in the table that contains the proper sum or product.

Example 2.3

A table of 1024 or 2^{10} entries can be used for moduli up to 32, or 2^5; i.e., if x_i and y_i are 5-bit arguments, then their concatenated 10-bit value forms an address into a table of results.



In this case, addition/subtraction and multiplication are accomplished in one access time to the table. Note that since access time is a function of table size and since the table size grows at 2^{2n} (number of bits to represent a number), residue arithmetic has a considerable advantage over conventional representation in its use of table look-up techniques. ♢

2.1.5 Conversion To and From Residue Representation

Conversion from an ordinary weighted number representation into a residue representation is conceptually simple—but implementations tend to be somewhat less obvious.

Conceptually, we could just divide the number to be converted by each of the respective moduli, and the remainders would form the residues. This process is usually too slow, however, and the integer to be converted is decomposed (as in the $2^k - 1$ case) and its components are converted and summed modulo the respective base. Thus, an integer A can be represented in familiar weighted positional notation:

$$A = \sum_{i=0}^{n} A_i R^{n-i},$$

where R = radix
A_i = the value of the ith position

It is decomposed with respect to radix position, or pairs of positions, simply by the ordered configuration of the digits.

In the usual case, the radix and the modular base are relatively prime, and for single-position conversion we would have:

$$x_{ji} = A_i R^{n-i} \, mod \, m_j,$$

where x_{ji} is the $i$thcomponent of the m_j residue of A, and then x_j (the residue of $A\,mod\ m_j$) is

$$x_j = \left(\sum_i x_{ji}\right)\ mod\ m_j.$$

The process can be quickly implemented. Since

$$x_{ji} = (A_i\,mod\ m_j \cdot R^{n-i}\,mod\ m_j)\,mod\ m_j,$$

the R^{n-i} $mod\ m_j$ term is precomputed and included in a table that maps A_i into x_{ji}. Thus, x_{ji} is derived from A_i in a single table look-up.

Example 2.4

Compute the residue mod 7 of the radix 10 integer 826.

$$\begin{aligned} 826 &= 8 \times 100 + 2 \times 10 + 6 \\ &= A_0 \times 10^2 + A_1 \times 10 + A_2 \end{aligned}$$

Now,

$$\begin{aligned} 100\,mod\ 7 &= 2 \\ 10\,mod\ 7 &= 3. \end{aligned}$$

Thus, we have the following tables:

A_0	X_{j0}	A_1	X_{j1}	A_2	X_{j2}
0	0	0	0	0	0
1	2	1	3	1	1
2	4	2	6	2	2
3	6	3	2	3	3
4	1	4	5	4	4
5	3	5	1	5	5
6	5	6	4	6	6
7	0	7	0	7	0
8	2	8	3	8	1
9	4	9	6	9	2

$826\,mod\ 7 = (2+6+6)\,mod\ 7 = 0$. Larger tables reduce the number of additions required, and thus may improve the speed of conversion. $\diamondsuit$

There is an important special case of conversion into a residue system: converting a $mod\ \ 2^n$ number into a residue representation $mod\ \ 2^k$ or $mod\ \ 2^k - 1$. This case is important because of the previously mentioned coding efficiency with these moduli, and because $mod\ \ 2^n$ numbers arise from arithmetic operations using conventional binary type logic.

The conversion process from a binary representation (actually, a residue mod 2^n) to a residue of either 2^k or $2^k - 1\,(n > k)$ is as follows: partition the n bits into m digits of size k bits; that is, $m = \lceil \frac{n}{k} \rceil$. Then a binary number $X\,mod\ 2^n$ is:

$$X_{\text{base 2}} = X_{n-1}\,2^{n-1} + X_{n-2}\,2^{n-2} + \cdots + X_0,$$

where X_i has value 0 or 1, and can be rewritten as:

$$X_{\text{base }2^{\text{k}}} = X_{m-1}\,(2^k)^{m-1} + X_{m-2}\,(2^k)^{m-2} + \cdots + X_0,$$

where X_i has values $\{0, 1, \ldots 2^k - 1\}$. This is a simple regrouping of digits. For example, consider a binary 24-bit number arranged in eight 3-bit groups.

$$X_{\text{base 2}} = 101\ 011\ 111\ 010\ 110\ 011\ 110\ 000.$$

This may be rewritten in octal ($k = 3$) as $\lceil \frac{n}{k} \rceil = \lceil \frac{24}{3} \rceil$ digits:

$$X_{\text{base 8}} = 5\ 3\ 7\ 2\ 6\ 3\ 6\ 0.$$

The residue $X\,mod\ 2^k = X_0$ (the least significant k bits), since all other digits in the representation are $0\,mod\ 2^k$, $(k = 3)$.

Now the residue of $X_{\text{base }2^{\text{k}}}\ mod\ (2^k - 1)$ can be computed directly from the mod 2^k representation. If X is a base 2^k number with n digits $(X_{n-1} \ldots X_0)$, and X_i is its ith digit:

$$X\ mod\ (2^k - 1) = \left(\sum_{i=0}^{n-1} X_i (2^k)^i\ mod\ (2^k - 1) \right)\ mod\ (2^k - 1).$$

For $X_0\,mod\ (2^k - 1)$, the residue is the value X_0 for all digit values except $X_0 = 2^k - 1$, where the residue is 0. Similarly, for $X_1 2^k\ mod\ 2^k - 1$, the residue is X_1 (1 for each 2^k multiple) unless X_1 itself is $2^k - 1$, in which case the residue is 0. For $X_i(2^k)^i\ mod\ 2^k - 1$, the residue $= X_i$, where $X_i \neq 2^k - 1$ and the residue $= 0$ if $X_i = 2^k - 1$. This is the familiar process of "casting-out" $(b - 1)$. In the previous example (X in octal),

$$X = 5\,3\,7\,2\,6\,3\,6\,0$$

and

$$x = X\ mod\ 7 = (5 + 3 + 0 + 2 + 6 + 3 + 6 + 0)\ mod\ 7.$$

Now, pair-sum mod 7 can be directly computed from a mod 8 adder by recognizing three cases:

1. $a + b < 2^k - 1$, that is, $a + b < 7$; then $(a + b)\ mod\ 7 = a + b\ mod\ 8 = a + b$.
2. $(a + b) = 2^k - 1$, $a + b = 7$; then $(a + b)\ mod\ 7 = 0$—that is, cast out 7's.
3. $a + b > 2^k - 1$, that is, $a + b > 7$; then $(a + b)\ mod\ 7 = a + b + 1$. A carryout occurs in a mod 8 adder. This end-around carry must be added to the mod 8 sum. If $a + b > 7$, then $a + b + 1 = (a + b)\ mod\ 7$, i.e., we use end-around carry.

In our example, $x = X\ mod\ 7 = (5 + 3 + 0 + 2 + 6 + 3 + 6 + 0)\ mod\ 7$.

	$\underbrace{5+3}$	$\underbrace{0+2}$	$\underbrace{6+3}$	$\underbrace{6+0}$
octal	10	2	11	6
mod 7	$1+0=1$	2	$1+1=2$	6
octal	$1+2=3$		$2+6=10$	
mod 7	3		$1+0=1$	
octal		$3+1=4$		
mod 7		4		
and $x=4$				

Conversion from residue representation is conceptually more difficult; however, the implementation is also straightforward [37].

First, the integer that corresponds to the residue representation that has a "1" in the $j$thresidue position and zero for all other residues is designated the *weight* of the jth residue, w_j. The ordering of the residues (the "j"s) is unimportant; however, since they are ordered, only one integer (mod the product of relatively prime moduli) will have a residue representation of 0, 0, 1, 0 ...0. That is, it would have a zero residue for all positions $\neq j$ and a residue $= 1$ at j. Now the problem is to scale the weighted sum of the residues up to the integer representation modulo M, the product of the relatively prime moduli. By construction of the weights, w_j, the product

$$\sum_k (x_j \cdot w_j)\, mod\ m_k = x_j,$$

since w_j is a multiple of all m_k $(k \neq j)$ and $(x_j \cdot w_j)$ mod $m_j = X\, mod\ m_j$ for all j. Thus, to recover the integer X from its residue representation, all we do is to sum the weighted residue modulo M:

$$X\, mod\ M = \left(\sum (w_j \cdot s_j)\right) mod\ M.$$

Example 2.5

Suppose we wish to encode integers with the relatively prime moduli 4 and 5. The product (M) is 20. Thus, we encode integers 0 through 19 in residue representation as follows:

	Residues	
X	x mod 4	x mod 5
0	0	0
1	1	1
2	2	2
3	3	3
4	0	4
5	1	0
6	2	1
7	3	2
8	0	3
9	1	4
10	2	0
11	3	1
12	0	2
13	1	3
14	2	4
15	3	0
16	0	1
17	1	2
18	2	3
19	3	4

where $w_1 =$ Value of X for which residue representation is $[1, 0] = 5$
$w_2 = [0, 1] = 16$.

Suppose we encode two integers, 5 and 13, in this representation:

$$\begin{aligned} 5 &= [1, 0] \\ 13 &= [1, 3] \end{aligned}$$

If we now wished their sum, we would get:

$$\begin{array}{r} [1,\ 0] \\ +\ [1,\ 3] \\ \hline [2,\ 3] \end{array}$$

To convert this to integer representation:

$$\begin{aligned} (x_1 w_1 + x_2 w_2)\ mod\ 20 &= X \\ (2 \cdot 5 + 3 \cdot 16)\ mod\ 20 &= 18. \end{aligned}$$

◊

2.1.6 Using the Residue Number System

In the past, the importance of the residue system lay in its theoretic significance rather than in its fast arithmetic capability. While multiplication is straightforward, division is not, and comparisons are quite complex. This, coupled with conversion problems, has limited the applicability of residue arithmetic. With the availability of powerful arithmetic technology, this may change for suitable algorithms and applications. In any event, it remains an important theoretic system, as we shall see when determining the computational time bounds for arithmetic operations.

Another important application of residue arithmetic is error checking. If, in an n-bit binary system:

$$\begin{array}{r} a \ mod \ 2^n \\ +b \ mod \ 2^n \\ \hline c \ mod \ 2^n \end{array}$$

then it also follows that:

$$\begin{array}{r} a \ mod \ (2^k - 1) \\ +b \ mod \ (2^k - 1) \\ \hline c \ mod \ (2^k - 1) \end{array}$$

Since 2^n and $2^k - 1$ are relatively prime, a small k-bit adder ($n \gg k$) can be used to check the operation of the n-bit adder. In practice, $k = 2$, 3, 4 is most commonly used. The larger k's are more expensive, but since they provide more unique representations, they afford a more comprehensive check of the arithmetic. For more information on using residue arithmetic in error checking, see the paper by Watson and Hastings [45].

2.2 The Limits of Fast Arithmetic

2.2.1 Background

The purpose of this section is to present the theoretic bounds on speed of arithmetic operations, so they can be compared against the state of art in arithmetic algorithms. These bounds serve as a yardstick to measure practical results, and provide a clear understanding of how much more speed improvement can be obtained.

2.2.2 Speed in Terms of Gate Delays

The execution speed of an arithmetic operation is a function of two factors. One is the circuit technology, and the other is the algorithm used. It can be confusing to discuss both factors simultaneously; e.g., a ripple carry adder implemented in ECL technology may be faster than a carry-look-ahead adder implemented in CMOS. In this section, we are interested only in the algorithm and not in the technology; therefore, the speed of the algorithms will be expressed in terms of gate delays. Using this approach, the carry-look-ahead adder is faster than the ripple carry adder. Simplistically translating gate delays for a given technology to actual speed is done by multiplying the gate delays by the gate speed.

2.2.3 The (r, d) Circuit Model

Much of the original work to determine a minimum bound on arithmetic speed was done by Winograd [46, 47]. In his model, the speed (in gate delays) of any logic and arithmetic operation is a function of three items:

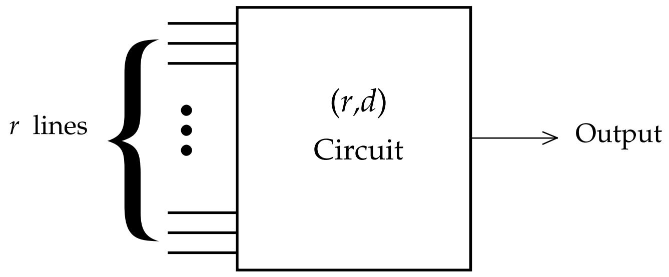


Figure 2.1: The (r, d) circuit.

1. Number of digits in each operand $= n$.
2. Fan-in of the gate (circuit) $= r =$ maximum number of logic inputs or arguments for a logic element.
3. The radix of the arithmetic $= d =$ number of truth values in the logic system.

Definition: An (r, d) circuit is a d-valued logic circuit in which each element has fan-in of at most r, and can compute any r-argument d-valued logic function in unit time.

In any practical technology, logic path delay depends upon many factors: the number of gates (circuits) that must be serially encountered before a decision can be made, the logic capability of each circuit, cumulative distance among all such serial members of a logic path, the electrical signal propagation time of the medium per unit distance, etc. In many high-speed logic implementations, especially those using ECL, the majority of total logic path delay is frequently attributable to delay external to logic gates. Thus, a comprehensive model of performance would have to include technology, distance, geography, and layout, as well as the electrical and logical capabilities of a gate. Clearly, the inclusion of all these variables makes a general model of arithmetic performance infeasible. Winograd's (r, d) model of a logic gate is idealized in many ways:

1. There is zero propagation delay between logic blocks.
2. The output of any logic block may go to any number of other logic blocks without affecting the delay; i.e., the model is *fan-out* independent. The fan-out of a gate refers to its ability to drive from output to input a number of other similar gates. Practically speaking, any gate has a maximum limit on the number of circuits it may communicate with based on electrical considerations. Also, as additional loads are added to a circuit, its delay is adversely affected.
3. The (r, d) circuit can perform any logical decision in a unit delay—more comments on this below.
4. Finally, the delay in, and indeed the feasibility of, implementations are frequently affected by mechanical considerations such as the ability to connect a particular circuit module to another, or the number of connectors through which such an electrical path might be established. These, of course, are ignored in the (r, d) model.

Despite these limitations, the (r, d) model serves as a useful first approximation in the analysis of the delay/performance of arithmetic algorithms in most technologies. The effects of propagation delay, fan-out, etc., are merely averaged out over all blocks to give an initial estimate as to the delay in a particular logic path. Thus, in a particular technology such as ECL, the basic delay within a block may be one nanosecond; but the effect of delay, including average path lengths, line loading effects, fan-out, etc., might be closer to 3 and 3.5 nanoseconds. Still, the number of blocks encountered between functional input and final result is an important and primary determinant (again, for most technologies) in determining speed.

The (r, d) model is a fan-in limited model, the number of inputs to a logical gate is limited at r inputs, each gate has one output, and all gates take a unit delay time (given valid inputs) to establish an output. The model allows for multivalued logic, where d is the number of values in the logic system. The model further assumes that any logic decision capable of being performed within an r inputm d-valued truth system is available in this unit time. This is an important premise. For example, in a 2-input binary logic system ($r = 2$, $d = 2$) there are 16 distinct logic functions (AND, OR, NOT, NOR, NAND, EQUALITY, IMPLICATION, etc.). In fact, there are in general d^{d^r} distinct logic functions in a general (r, d) logic system. In any practical logic system, only a small subset of these are available. These are chosen in such a way as to be *functionally complete*, i.e., able to generate any of the other logic expressions in the system. However, the functionally complete set in general will not perform a required arbitrary logic function in unit delay, e.g. NOR's implementing EXCLUSIVE OR requires two unit delays. Thus, the (r, d) circuit is a lower bound on a practical realization. What we will discover in later chapters is that familiar logic subsets (e.g., NOR) can by themselves come quite close to the performance predicted by the (r, d) model.

2.2.4 First Approximation to the Lower Bound

Spira [32] has shown that if a d-valued output f is a function of all n arguments (d-valued inputs), then t, the number of (r, d) delays, is:

$$t \geq \lceil \log_r n \rceil$$

in units of (r, d) circuit delay.

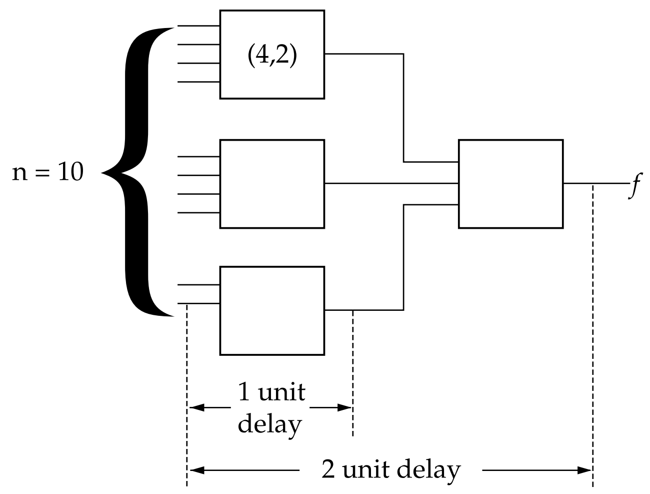
Example 2.6 ($n = 10$, $r = 4$, $d = 2$)

$$\lceil \log_r n \rceil = \lceil \log_4 10 \rceil = \lceil 1.65 \rceil = 2$$

Proof: Spira's bound can be proved by induction and follows from the definition of the (r, d) circuit. The (r, d) circuit has a single output and r inputs; thus, a single level ($t = 1$) has r inputs. Let f_t designate a circuit with n inputs and t units of delay.

Consider the case of unit delay, i.e., $t = 1$. Since the fan-in in a unit block is r, then if the number of inputs n is less than or equal to r:

$$1 \geq \lceil \log_r n \rceil,$$



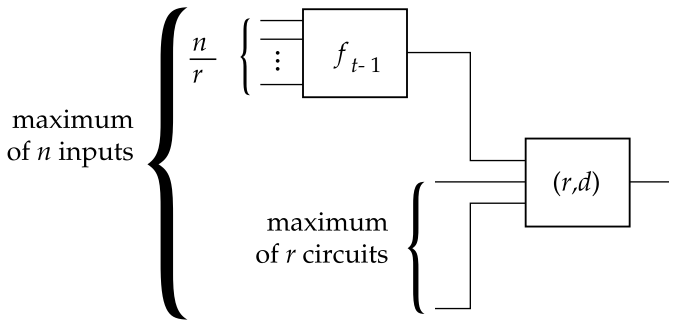


Figure 2.2: The (r, d) network.

since we have to have at least one gate to define the function f. Now suppose Spira's bound is correct for delays in a $t-1$ circuit (f_{t-1}). Let us find the resulting delay in the network (see Figure 2.2) for f_t. We are given that f_{t-1} is a function of n/r inputs. Now we have:

$$t - 1 \geq \lceil \log_r(n/r) \rceil \ = \lceil \log_r(n) - log_r(r) \rceil = \lceil \log_r(n) \rceil - 1$$

and

$$t \geq \lceil \log_r(n) \rceil .$$

This proves the bound.

Now we can derive the lower bound for addition in the residue number system. $\diamondsuit$

2.2.5 Spira's Bound Applied to Residue Arithmetic (Winograd's Bound)

The time for addition using (r, d) circuits and the residue system is at least:

$$t \geq \lceil \log_r 2 \lceil \log_d \alpha(N) \rceil \rceil ,$$

where $\alpha(N)$ is the number of elements representable by the largest of the relatively prime moduli.

Clearly, since arithmetic is carry independent between the various moduli, we only need concern ourselves with the carry and propagation delay for the largest of the moduli. If this is N, then $\alpha(N)$ is the number of distinct numbers that this modulus can represent. Now $\log_d \alpha(N)$ is the number of d-valued lines required to represent a number for this modulus. Thus, an addition network for this modulus has $2\lceil \log_d \alpha(N) \rceil$ input lines. In the addition operation, a low order 1 can, for certain configurations of input line configurations, affect the most significant output line. The most significant output line then depends upon all input lines. Thus, by Spira's Bound, we have:

$$t \geq \left\lceil \log_r \underbrace{2 \underbrace{\lceil \log_d \alpha(N) \rceil}_{\text{number of digits}}}_{\text{input lines}} \right\rceil$$

Winograd's theorem is actually more general than the above, since it shows that the bound is valid not only for the residue arithmetic but for any arithmetic representation obeying group theoretic properties. In the general case of modular addition, the $\alpha(N)$ function needs more clarification. In modular arithmetic, we are operating with single arguments mod A^n. If A is prime, then $\alpha(N)$ is simply A^n, but if A is composite (i.e., not a prime), then $A = A_1 A_2 \ldots A_m$ and arithmetic can be decomposed into simultaneous operations mod A_1^n, mod A_2^n,... mod A_n^n. In this case, $\alpha(N)$ is A_i^n, where A_i is the largest element composing A.

For example, in decimal arithmetic, $A = 10^n = 2^n \cdot 5^n$ and independent pair arithmetic can be defined for A_2^n and A_5^n, limiting the carry computation to the largest modules; in this case $\alpha(10^n) = 5^n$.

Frequently, we are not interested in a bound for a particular modular system (say A^n), but in a tight lower bound for a residue system that has *at least* the capacity of A^n. We designate such a system ($> A^n$), since the product of its relatively prime moduli must exceed A^n.

Example 2.7

1. Modular representation:

$$\begin{array}{rl} \text{prime base} & \alpha(2^{12}) = 2^{12}, \\ \text{composite base} & \alpha(10^{12}) = 5^{12}; \end{array}$$

   Note: a composite base has multiple factors ($\neq 1$); e.g., $10 = 5.2$ is a composite base, while 2 is not composite.

2. Residue representation:

$$\alpha(> 2^{16}); \text{ using set } \{2^5, 2^5 - 1, 2^4 - 1, 2^3 - 1\} = 2^5.$$

Note that *minimum* $\alpha(> 2^k) = p_n$, where p_n is the nth prime in the product function defined as the smallest product of consecutive primes p_i, or *powers of primes*, that equal or exceed 2^{16}:

$$\prod_{i=1}^{n} p_i \geq 2^{16}.$$

The selection of moduli to minimize the α function is best illustrated by an example. $\diamondsuit$

Example 2.8

Suppose we wish to design a residue system that has $M \geq 2^{47}$, i.e., at least 2^{47} unique representations. We wish to minimize the largest factor of M, $\alpha(M)$, in order to assure fast arithmetic. If we simply selected the product of the primes, we would have:

$$2 \times 3 \times 5 \times 7 \times 11 \times 13 \times 17 \times 19 \times 23 \times 29 \times 31 \times 37 \times 41 > 2^{47};$$

that is, the $\alpha(> 2^{47})$ for this selection would be 41.

We can improve the α function by using powers of the lower order primes. Thus:

$$2^5 \times 3^3 \times 5^2 \times 7 \times 11 \times 13 \times 17 \times 19 \times 23 \times 29 \times 31 > 2^{47}.$$

Here, $\alpha(> 2^{47})$ is $2^5 = 32$. Thus, finding the minimum α function requires that before increasing the product (in the development of M) by the next larger prime, P_n, we check that there are no lower order primes, P_i, which when raised to their next integer power would not lie between P_{n-1} and P_n. That is, for each $i < n-1$ and x the next integer power is P_i.

$$P_{n-1} < P_i^x < P_n.$$

We use all such qualified P_i^x terms before introducing P_n into the product. $\diamondsuit$

2.2.6 Winograd's Lower Bound on Multiplication

Typical multiplication is simulated by successive add–shifts and takes n addition times, e.g., multiplication of 16-bit numbers (that can be added in 100 ns) takes 1.6 microseconds. Now, Winograd surprises us by saying that multiplication is not necessarily slower than addition! And, if this were not enough, multiplication can be even slightly faster than addition [4, 47].

Since multiplication is also a group operation involving two n-digit d-valued numbers (whose output is dependent on all inputs), the Spira bound applies.

$$t \geq \lceil \log_r 2n \rceil,$$

where $2n$ = the total number of d-valued input lines.

To see that multiplication can be performed at the same speed as addition, one need only consider multiplication by addition of the log representation of numbers: if $a * b = c$, then $\log a + \log b = \log c$.

Notice that in a log representation, fewer significant product bits are required than in the familiar linear weighted system. For example, $\log_2 16 = 4.0$ requires 4 bits (3, plus one after the binary point) instead of 5 bits, as $16.0 = 10000.0$ would require. Of course, log representations require subtraction (i.e., negative log) for numbers less than 1.0, and zero is a special case.

Since division in this representation is simply subtraction, the bound applies equally to multiplication and division. Also, for numbers represented with a composite modular base (i.e., A^n, where $A^n = A_1 \times A_2 \times \ldots \times A_n$), a set of log representations can be used. This coding of each base A number as an n-tuple $\{\log Ai;\ i = 1 \text{ to } n\}$ minimizes the length of the carry path by reducing the number of d-valued input lines required to represent a number.

As an analog to residue representation, numbers can be represented as composite powers of primes, and then multiplication is simply the addition of corresponding powers.

Example 2.9

12×20

$$\begin{aligned} 12 &= 2^2 \cdot 3^1 \cdot 5^0 \\ 20 &= 2^2 \cdot 3^0 \cdot 5^1 \\ \text{product } 240 &= 2^4 \cdot 3^1 \cdot 5^1 \end{aligned}$$

$12 \div 20$

$$\begin{aligned} 12 &= 2^2 \cdot 3^1 \cdot 5^0 \\ 20 &= 2^2 \cdot 3^0 \cdot 5^1 \\ 12/20 &= 2^0 \cdot 3^1 \cdot 5^{-1} = 3/5 \end{aligned}$$

Winograd formalizes this by defining $\beta(N)$ akin to the $\alpha(N)$ of addition and shows that for multiplication:

$$\boxed{t \geq \lceil \log_r 2 \lceil \log_d \beta(N) \rceil \rceil}$$

where

$$\boxed{\beta(N) < \alpha(N)}$$

The exact definition of $\beta(N)$ is more complex than $\alpha(N)$. Three cases are recognized:

Case 1: Binary radix ($N = 2^n$); $n \geq 3$

$$\beta(2^n) = 2^{n-2}$$

for Binary radix ($N = 2^n$); $n < 3$,

$$\begin{aligned} \beta(4) &= 2 \\ \beta(2) &= 1 \end{aligned}$$

Case 2: Prime radix ($N = p^n$); p a prime > 2

$$\begin{aligned} \beta(p^n) &= \max\left(p^{n-1}, \alpha(p-1)\right) \\ \text{e.g., } \beta(59) &= \alpha(58) = \alpha(29 \cdot 2) = 29 \end{aligned}$$

Case 3: Composite powers of primes ($N = p_1^{n_1} \cdot p_2^{n_2} \ldots p_m^{n_m}$)

$$\beta(N) = max\,(\beta(p_1^{n_1}), \ldots, \beta(p_i^{n_i}) \ldots)\,.$$

$\diamondsuit$

Example 2.10

1. $N = 2^{10} \quad \beta(2^{10}) = 2^8$.
2. $N = 5^{10} \quad \beta(5^{10}) = 5^9$.
3.

$$\begin{aligned} N = 10^{10} = 5^{10} \cdot 2^{10} &= \beta(5^{10}, 2^{10}) \\ &= max\left(\beta(5^{10}), \beta(2^{10})\right) \\ &= max(5^9, 2^8) \\ &= 5^9 \end{aligned}$$

In order to reach the lower bounds of addition or multiplication, it is necessary to use data representations that are nonstandard. By optimizing the representation for fast addition or multiplication, a variety of other operations will occur much slower. In particular, performing comparisons or calculating overflow are much more difficult and require additional hardware using this nonstandard representation. Winograd showed that both these functions require at least $\lceil \log_r(2\lceil \log_2 N \rceil) \rceil$ time units to compute [47]. In conventional binary notation, both of these functions can be easily implemented by making minor modifications to the adder. Hence, the type of data representation used must be decided from a broader perspective, and not based merely on the addition or multiplication speed. $\diamondsuit$

2.3 Modeling of ROM Speed in Gate Delays

As an alternative to computing sums or products each time the arguments are available, one could consider simply storing all possible results in a table. We then could use the arguments to look up (address) the answer, as shown in the example on page 56.

Would such a scheme lead to even faster arithmetic, i.e., better than the (r, d) bound? The answer is probably not, since the table size grows rapidly as the number of argument digits, n, increases. For b-based arithmetic, there are b^{2n} required entries. Access delay naturally is a function of table size.

Modeling this delay is not the same as finding a lower time bound, however. In ROM's as well as many storage technologies, the access delay is a function of many physical parameters. What we present here is a simple model of access delay as an approximation to the access time.

We start by a simple model of 16×1 ROM (Figure 2.3):

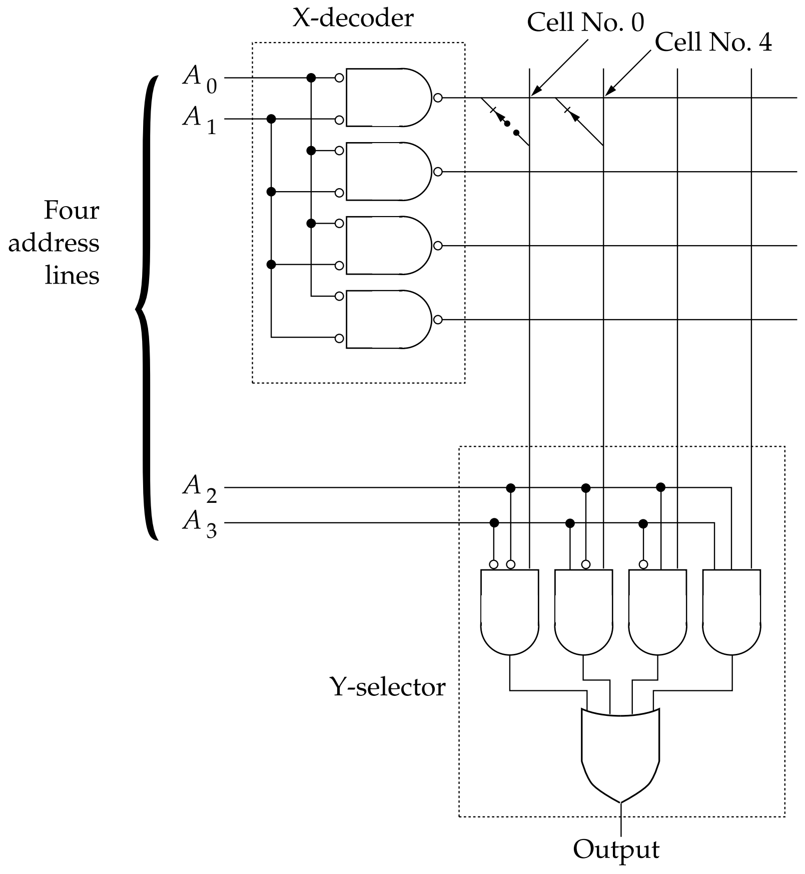


Figure 2.3: ROM model.

This ROM is made of 16 cells which store information by having optional diode connections at each row and column intersection. For example, in the above figure, cell #0($A_3A_2A_1A_0$ = 0000) stores one, and cell #4 stores a zero. The delay of the ROM is a combination of the X decoder, the diode matrix, and the Y selector. In the above case (for fan-in = 4), the ROM delay is made of four gates (assuming the diode matrix is one gate delay). In general, a ROM with L address lines has the following delays:

$$\begin{aligned} X\text{-decode} &= \left\lceil \log_r \left(\frac{\mathrm{L}}{2}\right) \right\rceil \\ \text{Diode matrix} &= 1 \\ Y\text{-selector} &= \left\lceil \log_r \left(\frac{\mathrm{L}}{2}+1\right) \right\rceil + \left\lceil \log_r 2^{\frac{\mathrm{L}}{2}} \right\rceil \end{aligned}$$

Half of the address lines ($\frac{\mathrm{L}}{2}$) are decoded in the X dimension, and according to Spira's bound the associated delay is $\lceil \log_r(\frac{\mathrm{L}}{2}) \rceil$.

In the Y-selector delay, the fan-in to each gate is composed of the $\frac{\mathrm{L}}{2}$ address lines plus a single input from the ROM array. These gates must, in turn, be multiplexed to arrive at a final result. As there are $2^{\frac{L}{2}}$ array outputs, there are $\lceil \log_r 2^{\frac{L}{2}} \rceil$ stages of delay, again by the Spira argument.

Actually, since only the ROM input to the Y-selector is critical, an improved configuration can be realized. The input to the Y-selector from the ROM is brought down to a single gate. The other input to this gate is the decoded Y-selection. Now the Y-selection delay is increased by one gate delay, but this is no worse than the X-decode plus the diode matrix delay. Thus:

$$\text{Unoverlapped Y-selector } = 1 + \left\lceil \log_r 2^{\frac{L}{2}} \right\rceil .$$

In the special case where $\lceil \log_r L/2+1 \rceil = 1$, the gate delays should be $1+0=1$, as the selector and AND gate can be integrated.

Example 2.11

For $1K$ word, ROM $L = 10$, and if we assume $r = 5$, then:

$$\begin{aligned} X\text{-decode} &= 1 \\ \text{diode matrix} &= 1 \\ Y\text{-selector} &= 1+3=4 \\ \text{total} &= 6 \text{ gate delays} \end{aligned}$$

When the ROM is used as a binary operator on n-bit numbers, the preceding formula can be expressed as a function of n, where $n = \frac{L}{2}$:

$$\text{ROM delay} = 2 + \lceil \log_r n \rceil + \lceil \log_r 2^n \rceil .$$

In many ways, this ROM delay points out the weakness of the (r, d) circuit model. In practical use of LSI ROM inplementations, the delay equation above is conservative when normalized to the gate delay in the same technology. The (r, d) model gives no "credit" to the ROM for its density, regular implementation structure, limited fan-out requirements, etc. $\diamondsuit$

2.4 Additional Readings

The two classic works in the development of residue arithmetic are by Garner [15] and Szabo and Tanaka [39]. They both are recommended to the serious student.

A readable, complete proof of Winograd's addition bound is found in Stone [37], a book that is also valuable for its introduction to residue arithmetic.

2.5 Summary

Alternate representation techniques exist using multiple moduli. These are called residue systems, with the principle advantage of allowing the designer use of small independent operands for arithmetic. Thus, a multitude of these smaller arithmetic operations can be performed simultaneously with a potential speed advantage. As we will see in later chapters, the speed advantage is usually limited to about a factor of 2 to 1 over more conventional representations and techniques. Thus, the difficulty in performing operations such as comparison and overflow detection limits the general purpose applicability of the residue representation approach. Of course, where special purpose applications involve only the basic add–multiply, serious consideration could be given to this approach.

Winograd's bound, while limited in applicability by the (r, d) model, is an important and fundamental limitation to arithmetic speed.

2.6 Exercises

1. Using residues of the form 2^k and $2^k - 1$, create an efficient residue system to include the range ± 32. Develop all tables and then perform the operation $-3 \times 2 + 7$.

2. The residue system is used to span the range of 0 to 10,000. What is the best set that includes the smallest maximum modulus (i.e., $\alpha(N)$)?

   (a) If any integer modulus is permitted.

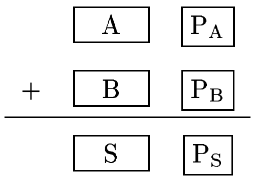
   (b) If moduli only of the form 2^k or $2^k - 1$ are allowed.

3. Repeat the above problem, if the range is to be $\pm 8,192$.

4. Analyze the use of an excess code as a method of representing both positive and negative numbers in a residue system.

5. Suppose two m bit numbers, A and B, are to be added and the sum checked using an even parity check on the sum S. A parity check on the sum is proposed for error detection.

   (a) Show that this scheme cannot be used in general to detect errors in *addition* (in the sum); i.e., $P(P_{\mathrm{A}} + P_{\mathrm{B}}) = P_{\mathrm{S}}$. The parity on the sum of P_{A} and P_{B} is compared against the parity of S.



   (b) Describe an n-bit check (i.e., P_A, P_B, and P_S, each n bits) so that arithmetic errors $(+, -, *)$ can be detected in the previous problem.

   (c) Find the probability of an undetected error in this system, where this probability is defined as:

$$\frac{\text{Number of valid representations}}{\text{Total number of representations}}.$$

(d) Devise an alternative scheme that will provide a complete check on the sum using parity. This system may use logic on the individual bit sum and carry signals to complete the check.

6. In Section 2.2.5, the optimum decomposition of prime factors was derived for $M \geq 2^{47}$. Following the 32 term, find the next seven factors (either a new prime or a power of prime) to form M' to be used in enlarging M. What is the new M' (approximately) and the new $\alpha(M')$?

7. If $r = 4$, $d = 2$, and M and M' are defined in Problem 6, find:

   (a) Lower bound on addition.

   (b) Lower bound on multiplication.

   (c) Number of equivalent gate delays in using a ROM implementation of addition or multiplication.

8. It is desired to perform the computation $z = \frac{1}{x} + y$ in hardware as fast as possible. If x and y are 8 bits, evaluate the number of gate delays ($r = 4$). Assume x, y are fractions ($.5 \leq a,\ b < 1$):

   (a) A single table look-up is used to evaluate z.

   (b) A table look-up is used to find $\frac{1}{x}$, then the result is added to an adder with gate delays $= 4\lceil \log_r 2n \rceil$.

   (c) If x and y are n-bit numbers, for what values of n does (a) have superior in-gate delay to (b)?
   Hint: Ignore ceiling function effects in your evaluation.

9. It has been observed that one can check a list of sums by comparing the single digit sum of each of the digits, e.g:

$$\begin{array}{rrlr}
 & 374 & 3+7+4=14; & 1+4=5 \\
 & 281 & 2+8+1=11; & 1+1=2 \\
 & \underline{523} & 5+2+3=10; & \underline{1+0=1} \\
 & 1178 & & 5+2+1 \\
1+1+7+8=17;\ 1+7=8 & & \longleftarrow \textit{check} \longrightarrow & =8
\end{array}$$

   (a) Does this always work?

   (b) Why? Explain in detail. If (a) is yes, prove it. If (a) is no, show counterexample and develop a scheme that will work.

10. What is the range of signed integers that can be represented in the 32, 31, 15 residue number system? Show how the following operation would be performed in this residue system.

$$\begin{array}{r}
123 \\
-\ 283 \\
\hline
\end{array}$$

11. It has been suggested that a "cast-out 8's" check can be used to check decimal addition. It goes like this:

> Find a check digit for each operand by summing the digits of a number. If the result contains multiple digits, sum them until we are reduced to a single digit. If anywhere along the way we encounter an '8,' discard the '8' and *subtract 1*! If we encounter a '9,' ignore it (i.e., treat it as '0').

The sum of the check digits then will equal the check digit of the sum.

E.g:

$$\begin{array}{rrrrrrcll}
 & 3 & 4 & 8 & 3 & 1 & = & 3+4-1+3+1=10=1+0 & =1 \\
 & 8 & 8 & 7 & 2 & 1 & = & -1-1+7+2+1=8 & =-1 \\
\hline
1 & 2 & 3 & 5 & 5 & 2 & & & 0 \\
\multicolumn{6}{r}{1+2+3+5+5+2} & = & 18=1-1 & =0
\end{array}$$

Does this always work? Prove or show a counterexample.

12. Yet *another* sum checking scheme has been proposed!

It goes like this: Find the check digits by adding *pairs* of digits together, reducing to a final pair. Then subtract the leading digit from the unit digit. If the result is negative ($\neq -1$), recomplement (i.e., add 10) and then add "1." If -1, leave it alone. Always reduce to a single digit, either -1, 0, or a positive digit.

E.g.:

$$\begin{array}{lcl}
0\,3\,4\,8\,3\,1 = 03 + 48 + 31 = 82 = -6 = & +5 & \\
\underline{0\,8\,8\,7\,2\,1} = 08 + 87 + 21 = 116 = 17 = & \underline{+6} & \\
 & 1\;\;1 & =0 \\
1\,2\,3\,5\,5\,2 & \overline{} & \\
12 + 35 + 52 = 99 = 0 & = \;\; 0 &
\end{array}$$

How about this one? Will it always work? Prove, or show a counterexample.